art and *LArSoft* course schedule

Draft version 1

Contents

1 Monday **2**

1.1 Session 1: Basics of C++ 2

1.2 Session 2: Basics of objects 2

1.3 Session 3: Basic data structures 3

1.4 Session 4: Framework introduction 3

1.5 Session 5: Setup for using *art* 3

1.6 Session 6: Setting up for development of experiment code 3

2 Tuesday **4**

2.1 Session 7: More module interface 4

2.2 Session 8: Details of module configuration 4

2.3 Session 9: Multiple instances a module 4

2.4 Session 10: Using existing data products 4

2.5 Session 11: Making histograms. 5

2.6 Session 12: Running multiple modules 5

3 Wednesday **5**

3.1 Session 13: Creating a *Producer* 6

3.2 Session 14: Inventing a new data product 6

3.3 Session 15: Controlling output 6

3.4 Session 16: Introducing iterative algorithm development. 6

3.5 Session 17: Writing a new algorithm. 6

3.6 Session 18: Using the algorithm in a producer 7

4 Thursday **7**

4.1 Session 19: Some additional *art* facilities 7

4.2 Session 20: Using *Assns* and smart query objects 7

4.3 Session 21: Creating *Assns* 7

4.4 Session 22: Code design issues. 8

4.5 Session 23: Good *art* workflow 8

4.6 Session 24: Reorganizing a supermodule. 8

5 Friday **8**

5.1 An introduction to *LArSoft* 8

5.2 Hands-on refactoring of code 9

Note: almost all "talks" are scheduled to be 20 minutes. This should be an upper bound; if only 15 minutes are needed, the extra time can go to discussion at the end.

1 Monday

The goal of Monday morning is to get people up to speed on some critical parts of C++ upon which we will rely for the entire course. We will introduce some of the basics of good coding practice. Some registrants for the course might not need this material. The material is designed that such students can skip this session.

The goal of Monday afternoon is to introduce people to the parts of the framework, and to the environment in which framework programs are run.

1.1 Session 1: Basics of C++

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

Pointers and references. Function calling and argument passing (reference and value); function return values. Compiling and linking; creating dynamic libraries. Understanding the difference between compilation and link errors.

1.2 Session 2: Basics of objects

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

Object lifetimes. RAII, *shared_ptr* and *unique_ptr*. Avoiding use of *new* and *delete*; avoiding use of *static*.

1.3 Session 3: Basic data structures

1 hour: 20 minutes talk, 30 minutes working, 10 minutes wrap-up.

Introduction to correct use of *std* library components *array*, *vector*, *map*, and *unordered_map*. Performance characteristics; when to use each. Relying on move constructor for efficient use. Correct initialization.

1.4 Session 4: Framework introduction

30 minutes, talk.

Overview of the major features of the framework; what the framework does for you. Where your code goes into a framework program.

1.5 Session 5: Setup for using *art*

1.5 hours: 20 minutes talk, 45 minutes working, 25 minutes wrap-up.

Based on workbook exercise 1. Introduces the *art* runtime system.

1. UPS products
2. *setup* command
3. executing *art*, use of a few command-line flags.
4. Creating output files.
5. Basic introduction to FHiCL configuration file for *art*.

1.6 Session 6: Setting up for development of experiment code

2 hours: 20 minutes talk, 90 minutes work, 10 minutes wrap-up.

Based on workbook exercise 2, but we should use *mrb* rather than *cetbuildtools* directly.

Goes all the way from cloning a *git* repository with source code, using *mrb* for building what has been cloned, up to looking at a module, running *artmod* to create a module.

Exercises should include: 1. Fixing a canned compilation failure 2. Fixing a canned link failure 3. Fixing a failure to find a module

2 Tuesday

The goal of Tuesday is to familiarize people with the development environment, in the context of starting to analyze data (i.e. making histograms) in the *art* setting.

2.1 Session 7: More module interface

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Introduce re-establishing a development environment after logging out of a previous shell session. Introduce the full *EDAnalyzer* interface; introduction to *Run* and *SubRun* objects (but not as containers of products yet). Understanding the module lifecycle.

Based on workbook exercise 3.

2.2 Session 8: Details of module configuration

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

How to correctly write a module constructor. How to use *ParameterSet* objects. How to handle errors in constructors. Revisit RAII here, and use of compiler-generated member functions when possible. Revisit *override* keyword. Introduce *art* exception class.

Based on workbook exercise 4.

2.3 Session 9: Multiple instances a module

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Understanding module instances, and the concept of having more than one instance of the same module class in a workflow. Understanding the ordering guarantee (after producers and filters, but not ordered relative to each other) provided by *art* for analyzers (and output).

Based on workbook exercise 5.

2.4 Session 10: Using existing data products

30 minutes: 20 minutes talk, 10 minutes questions.

How to find data products, how to read headers to understand data products. Introduction to some good data product design practices. The examples should be from *LArSoft* data products.

2.5 Session 11: Making histograms.

2 hours: 20 minutes talk, 90 minutes work, 10 minutes questions.

Accessing event data products; introduction of details of module label and module type. Using *ValidHandle*; *Handle* is for special cases only. Simple configuration of analyzer modules; understanding module execution order guarantees provided by *art*.

Using data product classes, iterating through sequences, filling histograms. Introduction to *TFileService*: how to configure, where it writes output.

Using Standard Library algorithms and lambda expressions to loop over data products. Reasons to prefer SL algorithms to explicit loops; avoiding fencepost errors, avoiding trivial inefficiencies, avoiding needless copies and conversions.

Combining examples 6 and 7 of the workbook, modified to use *LArSoft* data products.

2.6 Session 12: Running multiple modules

1.5 hours: 20 minutes talk, 60 minutes work, 10 minutes questions.

Run a chain of reconstruction algorithms. Write an analyzer that compares data products from two different algorithms. Practice using *art::InputTag* to identify products; practice seeing how the configuration file determines how products are labeled. Introduction to paths, and the order in which producers are executed. How the framework avoid running the same configured module more than once.

At this point we have run out of workbook exercises to use.

---

3 Wednesday

The goal of Wednesday is to introduce people to writing producers, using good software development practices: iterative code development, writing modular code, and testing. We also introduce the creation of new data products.

3.1 Session 13: Creating a *Producer*

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Canonical form of a producer (get/do/put pattern). How *Event::put* works; how data products are labeled. How the FHiCL file controls each producer's module label and the process name. How to create an instance of a data product.

3.2 Session 14: Inventing a new data product

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Design guidelines for new data products following the *art* data product design guide.

Producing dictionaries for ROOT. *We are currently using ROOT5. We'll be in the process of moving to ROOT 6 around the time of the class. This may give us some problems to solve.*

Need a good motivating example for writing your own data product.

3.3 Session 15: Controlling output

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Introduce the output system's ability to do *event selection* and *product selection*. Detailed configuration of *RootOutput*. Keep/drop lists. Configuration and meaning of *drop on input*.

Controlling compression levels. Writing multiple output files.

3.4 Session 16: Introducing iterative algorithm development.

30 minutes: 20 minutes talk, 10 minutes questions.

Introduce the ideas behind iterative development of algorithms, and designing code for testability.

3.5 Session 17: Writing a new algorithm.

2 hours: 20 minutes talk, 1.5 hours work, 10 minutes wrap-up.

Implement in code an algorithm specified as part of the problem. This algorithm should be suitable for coding in a single module, not as several modules. Write unit tests for the algorithm. Use the build tools to build and run the tests. Illustrate how to work on the algorithm outside of the framework. Illustrate a clear separation between *unit testing* and *physics validation*.

3.6 Session 18: Using the algorithm in a producer

30 minutes: 5 minutes talk, 20 minutes work, 5 minutes wrap-up.

Use *artmod* to generate the skeleton of a producer. Insert the algorithm written in the previous session into the producer. Write the integration test that executes the producer.

4 Thursday

The goal of Thursday is to gain more experience in modularity. This includes the development of a set of algorithms (only need two) that need to be split up between modules. Should extend the previous day's work. Introduce people to some of the more advanced framework tools (association collections, *FindOne* and *FindMany* smart query objects, services), as well as the *art* command line.

4.1 Session 19: Some additional *art* facilities

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Explanation of full set of *art* command-line options. Use of configuration dumping facilities. Introduction to the most important of the standard services: *TimeTracker*, *MemoryTracker*, *Tracer*.

Any others? Do we need to deal with *RandomNumberGenerator* or can we agree that it is too much of a special-purpose thing for this class?

4.2 Session 20: Using *Assns* and smart query objects

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Use a canned module that creates some product from the output of Wednesday's producer, and *Assns* between that product and Wednesday's product.

Write an analyzer that uses *FindOne* or *FindMany* to obtain information to fill histograms. This doesn't directly require that users see the *Assns* object.

4.3 Session 21: Creating *Assns*

1 hour: 20 minutes talk, 30 minutes work, 10 minutes wrap-up.

Write a producer that does something like what the "canned producer" above does. Understand how to create *Assns* objects.

The Thursday afternoon session schedule is much more tentative than the previous days' schedules.

4.4 Session 22: Code design issues.

Using class design to help simplify code. Putting behavior into classes, rather than having code pull data members out of a class to manipulate them externally. The value of cohesive classes.

Function design. Abstractions, useful generalizations. Controlling dependencies.

4.5 Session 23: Good *art* workflow

Understanding how to break up a large task into several parts, each handled by a module. Understanding how to design a set of modules to work together (rather than inventing a new sub-framework within a single module).

This might be best as a design session in which no actual code is written.

4.6 Session 24: Reorganizing a supermodule.

Take an existing too-large module, with too much class state and a few (or one) overly large member functions, and plan how to break it up into pieces, using multiple modules each of which uses one or more algorithm objects to do their work. Redesign for testability.

This might be best as a design session in which no actual code is written.

5 Friday

This day we split into two tracks:

5.1 An introduction to *LArSoft*

An all-day session giving an introduction into the design philosophy and content of the *LArSoft* products.

Details need to come from the LArSoft team.

5.2 Hands-on refactoring of code

An all-day session in which small groups (or individuals) from each experiment works on refactoring existing code, from the experiment, to reflect what has been learned in the course. The TAs of the course will be present to help with the effort.

In the schedule should be a sufficient number of breaks, during which some topic that has been identified as being of possibly general interest is called out and discussed. The goal of this is to help each group be able to learn from the others' experience during the session.

The goal should be that, at the end of the day, the participants can submit some improved code back to their experiments' repositories.